Smart Contract Audit Report

February, 2023

DEFIMOON PROJECT

Audit and
Development

CONTACTS

defimoon.org
audit@defimoon.org
defimoon_org
defimoonorg
defimoon
defimoonorg

DEFIMOON

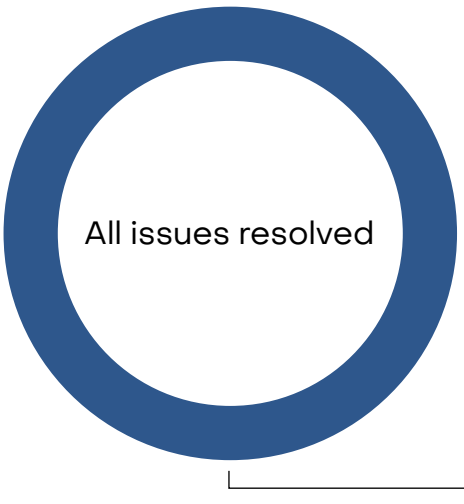
be secure

February 22th 2023

This audit report was prepared by Defimoon for Comtech-Gold

Audit information

Description	CGOController contracts.
Audited files	Goldtoken.sol, CGOController.sol
Timeline	13th February 2023-22th February 2023
Audited by	Daniil Rashin, Ilya Vaganov
Approved by	Artur Makhnach, Kirill Minyaev
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Manual Review
Specification	N/A
Docs quality	N/A
Source code	Github commit 9dac0fb
Network	Not specified
Status	Passed



	High Risk	A fatal vulnerability that can cause the loss of all Tokens / Funds.
	Medium Risk	A vulnerability that can cause the loss of some Tokens / Funds.
	Low Risk	A vulnerability which can cause the loss of protocol functionality.
	Informational	Non-security issues such as functionality, style, and convention.

Disclaimer

This audit is not financial, investment, or any other kind of advice and could be used for informational purposes only. This report is not a substitute for doing your own research and due diligence should always be paid in full to any project. Defimoon is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Defimoon has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Defimoon is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. Defimoon has no connection to the project other than the conduction of this audit and has no obligations other than to publish an objective report. Defimoon will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Defimoon assumes that the provided information and materials were not altered, suppressed, or misleading. This report is published by Defimoon, and Defimoon has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Defimoon. In instances where an auditor or team member has a personal connection with the audited project, that auditor or team member will be excluded from viewing or impacting any internal communication regarding the specific audit.

Audit Information

Defimoon utilizes both manual and automated auditing approach to cover the most ground possible. We begin with generic static analysis automated tools to quickly assess the overall state of the contract. We then move to a comprehensive manual code analysis, which enables us to find security flaws that automated tools would miss. Finally, we conduct an extensive unit testing to make sure contract behaves as expected under stress conditions.

In our decision making process we rely on finding located via the manual code inspection and testing. If an automated tool raises a possible vulnerability, we always investigate it further manually to make a final verdict. All our tests are run in a special test environment which matches the "real world" situations and we utilize exact copies of the published or provided contracts.

While conducting the audit, the Defimoon security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Defimoon assesses the risks and assigns a risk level to each section together with an explanatory comment.

Goldtoken Audit overview

Related findings:

- **DFM-1**

No tokens would be minted while deploying Goldtoken contract.

- **DFM-7**

Solidity version too old.

Goldtoken.sol

No major issues were found.

Simple token contract with mint, burn and blacklist features. There is a mint call in the constructor, but it would not mint the token in the current version. Functions have virtual modifier, so token may be inherited.

CGOController Audit overview

Related findings:

- **DFM-2**
No input check on setInitiator and setExecutor.

- **DFM-3**
State changes after token transfer in mint and burn functions.

- **DFM-4**
Redundant amount math.

- **DFM-5**
Unused minterWalletAddr.

- **DFM-6**
Indexing events.

- **DFM-7**
Solidity version too old.

- **DFM-8**
Contract functions could be declared external.

CGOController.sol

No major issues were found.

There are some some points where code may be improved, but generally there are no issues critically affecting core logic.

Summary of findings

According to the standard audit assessment, the audited solidity smart contracts are secure and ready for production, but there are few ways code may be improved.

ID	Description	Severity	Status
DFM-1	Minting 0 tokens	Informational	Acknowledged
DFM-2	Core addresses input check	Low risk	Resolved
DFM-3	Mint execution order	Informational	Resolved
DFM-4	Amount math improvement.	Informational	Acknowledged
DFM-5	Redundant minter wallet	Informational	Resolved
DFM-6	Indexed events	Informational	Acknowledged
DFM-7	Solidity version too old	Informational	Acknowledged
DFM-8	Functions could be declared external	Informational	Acknowledged

Application security checklist

Compiler errors	Passed
Possible delays in data delivery	Passed
Timestamp dependence	Passed
Integer Overflow and Underflow	Passed
Race Conditions and Reentrancy	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Private user data leaks	Passed
Malicious Events Log	Passed
Scoping and Declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Design Logic	Passed
Cross-function race conditions	Passed

Detailed Audit Information

Contract Programming

Solidity version not specified	Passed
Solidity version too old	Passed
Integer overflow/underflow	Passed
Function input parameters lack of check	Passed
Function input parameters check bypass	Passed
Function access control lacks management	Passed
Critical operation lacks event log	Passed
Human/contract checks bypass	Passed
Random number generation/use vulnerability	Passed
Fallback function misuse	Passed
Race condition	Passed
Logical vulnerability	Passed
Other programming issues	Passed

Code Specification

Visibility not explicitly declared	Passed
Variable storage location not explicitly declared	Passed
Use keywords/functions to be deprecated	Passed
Other code specification issues	Passed

Gas Optimization

Assert () misuse	Passed
High consumption ‘for/while’ loop	Passed
High consumption ‘storage’ storage	Passed
“Out of Gas” Attack	Passed
Public function could be external	Passed

Findings

DFM-1 « Minting 0 tokens»

Severity: Informational

Status: Acknowledged

Client's comment:
Token contract as its already live in the marketplace.

Description:
While deploying Goldtoken, call in constructor would mint 0 tokens since 0**18 is still 0. It would not severely affect project logic, but the call itself is redundant in this case.

`_mint(msg.sender, 10000000 * (0**uint256(decimals())))`
This call would mint 0 tokens to contract creator since 0**18 is 0.

Recommendation:
If mint call is required for inherited tokens, it may be passed via constructor arguments, as well as token name and symbol. Otherwise, redundant call should be removed.

DFM-2 «Core address input check»

Severity: Low Risk

Status: Resolved

Description: Functions setInitiatorAddr setExecutorAddr and setMinterWalletAddr may have zero-address value which would lead to loss of functionality while trying to address them.

Recommendation: Set variables in constructor & add zero-address checks to mentioned functions.

DFM-3 «Mint execution order»

Severity: Informational

Status: Resolved

Description:
Please, be aware of the mint function execution order. It is considered a safer approach to transfer tokens after all the required values are checked.

Recommendation: Move mint below all the requirements.

DFM-4 «Amount math improvement»

Severity: Informational

Status: Acknowledged

Client's comment:
Kept purposely to accept in ether denomination as per the project requirement to view it as human readable inside yplusvault multisig (transaction history - ABI view).

Description:
Amount parameter is treated differently in controller and token contracts. While token amount is treated as **wei**, which is the right way, CGOController performs multiplication inside its functions. It may lead to unexpected values being passed as arguments.

Recommendation:
Perform the 10**18 multiplication off-chain with **ethers.parseEther** for example.

DFM-5 «Redundant minter wallet»

Severity: Informational

Status: Resolved

Description:
minterWalletaddr is not used anywhere except setter functions.

Recommendation:
Remove unused variable.

DFM-6 «Indexed events»

Severity: Informational

Status: Acknowledged

Client's comment:
Stash the indexed event, unable to access event in our txn. handler after indexed implementation.

Description:
All the events parameters in CGOController miss **indexed** keyword, which may greatly increase contract API.

Recommendation:
Add **indexed** modifier to parameters that are commonly indexed.

DFM-7 «Solidity version too old»

Severity: Informational

Status: Acknowledged

Client's comment:
The contract as the token contract is already live in the market.

Description:
There were many bugs fixed since contracts version stated in pragma.

Recommendation:
Please, follow the official recommendation on choosing the version to use. Also it is much safer to use strict versioning like **0.8.9**

When deploying contracts, you should use the latest released version of Solidity. Apart from exceptional cases, only the latest version receives security fixes. Furthermore, breaking changes as well as new features are introduced regularly. We currently use a 0.y.z version number to indicate this fast pace of change.

Warning

Solidity recently released the 0.8.x version that introduced a lot of breaking changes. Make sure you read the full list.

DFM-8 «Functions could be declared external»

Severity: Informational

Status: Acknowledged

Client's comment:
The contract as the token contract is already live in the market.

Description:
All the functions in CGOController contract may be declared external which saves **gas**.

Recommendation:
Declare functions external instead of public.

Adherence to Best Practices

1. Consider amount values as wei.
2. Declare functions external where possible.
3. Make event parameters indexed.
4. Use strict recent solidity version.

Methodology

Manual Code Review

We prefer to work with a transparent process and make our reviews a collaborative effort. The goal of our security audits is to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, review open issue tickets, and investigate details other than the implementation.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system to make a final decision.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Appendix A — Finding Statuses

Resolved	Contracts were modified to permanently resolve the finding
Mitigated	The finding was resolved by other methods such as revoking contract ownership or updating the code to minimize the effect of the finding
Acknowledged	Project team is made aware of the finding
Open	The finding was not addressed